TELECOMUNICACIÓN

Campus Sur

POLITÉCNICA

PROYECTO FIN DE GRADO

TÍTULO: SYSTEM SUPPORTING PUBLIC TRANSPORT BASED ON GEOLOCATION

AUTOR: IVÁN GARRIDO GÓMEZ

TUTOR (o Director en su caso): PIOTR KOWALSKI

DEPARTAMENTO: DIATEL

TITULACIÓN: Grado en Ingeniería Telemática

V°B°

Miembros del Tribunal Calificador:

PRESIDENTE: ZBIGNIEW MROZEK

TUTOR: PIOTR KOWALSKI

SECRETARIO:

Fecha de lectura: 1 **de** julio **de** 2013

Calificación: A (MH)

El Secretario,

Resumen

El propósito de este proyecto de fin de Grado es el estudio y desarrollo de una aplicación basada en Android que proporcionará soporte y atención a los servicios de transporte público existentes en Cracovia, Polonia.

La principal funcionalidad del sistema será consultar la posición de un determinado autobús o tranvía y mostrar su ubicación con exactitud. Para lograr esto, necesitaremos tres fases de desarrollo.

En primer lugar, deberemos implementar un sistema que obtenga las coordenadas geográficas de los vehículos de transporte público en cada instante. A continuación, tendremos que registrar todos estos datos y almacenarlos en una base de datos en un servidor web. Por último, desarrollaremos un sistema cliente que realice consultas a tiempo real sobre estos datos almacenados, obteniendo la posición para una línea determinada y mostrando su ubicación con un marcador en el mapa.

Para hacer el seguimiento de los vehículos, sería necesario tener acceso a una API pública que nos proporcionase la posición registrada por los GPS que integran cada uno de ellos. Como esta API no existe actualmente para los servicios de autobús, y para los tranvías es de uso meramente privado, desarrollaremos una segunda aplicación en Android que hará las funciones del lado servidor. En ella podremos elegir mediante una simple interfaz el número de línea y un código específico que identificará a cada vehículo en particular (e.g. podemos tener 6 tranvías recorriendo la red al mismo tiempo para la línea 24). Esta aplicación obtendrá las coordenadas geográficas del teléfono móvil, lo cual incluye latitud, longitud y orientación a través del proveedor GPS.

De este modo, podremos realizar una simulación de como el sistema funcionará a tiempo real utilizando la aplicación servidora desde dentro de un tranvía o autobús y, al mismo tiempo, utilizando la aplicación cliente haciendo peticiones para mostrar la información de dicho tranvía.

El cliente, además, podrá consultar la ruta de cualquier línea sin necesidad de tener acceso a Internet. Almacenaremos las rutas y paradas de cada línea en la memoria del teléfono móvil utilizando ficheros XML debido al poco espacio que ocupan y a lo útil que resulta poder consultar un trayecto en cualquier momento, independientemente del acceso a la red.

El usuario también podrá consultar las tablas de horarios oficiales para cada línea. Aunque en este caso si será necesaria una conexión a Internet debido a que se realizará a través de la web oficial de MPK.

Para almacenar todas las coordenadas de cada vehículo en cada instante necesitaremos crear una base de datos en un servidor. Esto se resolverá mediante el uso de MYSQL y PHP. Se enviarán peticiones de tipo GET y POST a los servicios PHP que se encargarán de traducir y realizar la consulta correspondiente a la base de datos MYSQL.

Por último, gracias a todos los datos recogidos relativos a la posición de los vehículos de transporte público, podremos realizar algunas tareas de análisis. Comparando la hora exacta a la que los vehículos pasaron por cada parada y la hora a la que deberían haber pasado según los horarios oficiales, podremos descubrir fallos en estos. Seremos capaces de determinar si es un error puntual debido a factores externos (atascos, averías,…) o si por el contrario, es algo que ocurre muy a menudo y se debería corregir el horario oficial.

Abstract

The aim of this final Project (for University) is to develop an Android application that will provide support and feedback to the public transport services in Krakow. The main functionality of the system will be to track the position of a desired bus or tram line, and display its position on the map. To achieve this, we will need 3 stages: the first one will be to implement a system that sends the geographical position of the public transport vehicles, the second one will be to collect this data in a web server, and the last one will be to get the last location registered for the desired line and display it on the map.

For tracking the vehicles, we would need to have access to a public API that should be connected with each bus/tram GPS. As this doesn't exist in Krakow or at least is not available for public use, we will develop a second android application that will do the server side job. We will be able to choose in a simple interface the line number and a code letter to identify each vehicle (*e.g.* we can have 6 trams that belong to the line number 24 working at the same time). It will take the current mobile geolocation; this includes getting latitude, longitude and bearing from the GPS provider. Thus, we will be able to make a simulation of how the system works in real time by using the server app inside a tram and at the same time, using the client app and making requests to display the information of that tram.

The client will also be able to check the path of the desired line without internet access. We will store the path and stops for each line locally in the phone memory using xml files due to the few requirements of available space it needs and the usefulness of checking a path when needed.

This app will also offer the functionality of checking the timetable for the line, but in this case, it will link to the official Mpk website, so Internet access will be required.

For storing all the coordinates for each vehicle at every moment we will need to create a database on a server. We have decided that the easiest way is to use Mysql and PHP for the deployment of the service. We will send GET and POST requests to the php files and those files will make the according queries to our database.

Finally, based on all the collected data, we will be able to get some information about errors in the system of public transport timetables. We will check at what time a line was in each specific stop and compare it with the official timetable to find mistakes of time. We will determine if it is something that happens occasionally and related to external factors (e.g. traffic jams, breakdowns…) or if on the other hand, it is something that happens very often and the public transport timetables should be looked over and corrected.

Content

1. Definition of goals

The aim of this final Project (for University) is to develop an Android application that will provide support and feedback to the public transport services in Krakow. The main functionality of the system will be to track the position of a desired bus or tram line, and display its position on the map. To achieve this, we will need 3 stages: the first one will be to implement a system that sends the geographical position of the public transport vehicles, the second one will be to collect this data in a web server, and the last one will be to get the last location registered for the desired line and display it on the map.

For tracking the vehicles, we would need to have access to a public API that should be connected with each bus/tram GPS. As this doesn't exist in Krakow or at least is not available for public use, we will develop a second android application that will do the server side job. We will be able to choose in a simple interface the line number and a code letter to identify each vehicle (*e.g.* we can have 6 trams that belong to the line number 24 working at the same time). It will take the current mobile geolocation; this includes getting latitude, longitude and bearing from the GPS provider. Thus, we will be able to make a simulation of how the system works in real time by using the server app inside a tram and at the same time, using the client app and making requests to display the information of that tram.

The client will also be able to check the path of the desired line without internet access. We will store the path and stops for each line locally in the phone memory using xml files due to the few requirements of available space it needs and the usefulness of checking a path when needed.

This app will also offer the functionality of checking the timetable for the line, but in this case, it will link to the official Mpk website, so Internet access will be required.

For storing all the coordinates for each vehicle at every moment we will need to create a database on a server. We have decided that the easiest way is to use Mysql and PHP for the deployment of the service. We will send GET and POST requests to the php files and those files will make the according queries to our database.

Finally, based on all the collected data, we will be able to get some information about errors in the system of public transport timetables. We will check at what time a line was in each specific stop and compare it with the official timetable to find mistakes of time. We will determine if it is something that happens occasionally and related to external factors (e.g. traffic jams,

breakdowns…) or if on the other hand, it is something that happens very often and the public transport timetables should be looked over and corrected.

** We have used a Samsung Galaxy S2 with OS Android 4.1.2 for running all the tests.

2. Context and similar apps

Nowadays, the improvements of technology are changing our lifestyle and making easier or at least helping us with a lot of common and daily tasks in ways that we could not have imagined in the past. We can find these improvements in almost every area, but we will focus on mobile phones.

If we take a look at how our mobile phones looked like 10 years ago and we compare it with the current ones we realize that present ones are no longer just mobile phones but big services platforms.

Open Operative Systems like Android make possible to have an affordable terminal with a lot of functionalities like network access, GPS, compass, accelerometer, etc. Thanks to these available technologies and the creativity and dedication of thousands of developers and programmers we have a great market of all kind of applications that are just within our reach.

Thousands, millions of people use public transport every day, and we all know time is priceless, even more nowadays in big cities. We want to get home as fast as possible and without waiting. Some people realized about this, and today, we can find several applications on Android and IOS that give us the time that will take the desired bus or tram to be in our stop. Some public transport companies offer an API to access to all this data and make possible the creation of apps that will use this information for different purposes. For example, we can find these services in big cities all over the world such as London, Madrid, Paris… Nevertheless, this API is not available at the moment in Krakow.

We wanted to go further to make more visual and useful this kind of application. For that purpose, we have thought about using geolocation functionalities. If we have mobile phones with Internet Access and great maps API's like Google's one, why not showing in the map the current position of our desired public transport?

There are no apps with this real geolocation functionality open for public transport at the moment. However, some big companies use these apps in a private environment to measure and report all kind of statistics. For example, they can locate their fleet on a cartographic map, knowing speed and direction of each vehicle; check that the drivers obey traffic legislation and corporate requirements, analyze energy efficiency tracking fuel consumption, detect technical or mechanical failures, and etcetera.

As we cannot access this API at the moment (in case it exists in MPK Krakow), the aim of this application is to create all the infrastructure needed and make a

simulation of how it would work in case they open their API and grant public access to it. Thus, we will make a second application that will provide the GPS functionality and will act as if it was the real GPS of the tram were we will run our tests.

Nowadays, there can be some different opinions about geolocation in terms of privacy and security, that's why we are not getting as much advances as we could at this moment. But most of the experts related to technology and society think that these different opinions will become one and merge into the final acceptance that geolocation has more benefits than drawbacks, at least in some particular contexts (public transport will be one of them).

3. Server Side – Location Sender

3.1.Introduction

As we have previously briefly overviewed, due to the lack of access to a public API connected to the trams and buses GPS, we need to think about the different possibilities that we have left to offer a gps tracking functionality.

We have thought about 3 different solutions, choosing the most logical and efficient one. Before going deeper into the chosen solution, we will take a look at the other possibilities.

3.1.1. Algorithm based solution

The first solution consists in the creation of an algorithm based on the official public transport timetables provided by MPK. With this algorithm and the designed paths for each line, we would be able to make and estimation of the vehicle position for a specific time.

The function would check the chosen line number timetable at the time we are making our consult. If for example it is 19:41:15 (HH:mm:ss) and we look for the bus line number 100, we can get from the timetable database that at 19:39 it is on the stop X and at 19:43 it arrives to the stop Y. Knowing the coordinates of both stops, the algorithm would estimate the coordinates of the current position and would show it on the map, determining the time it will take to arrive to the station.

The reason why we didn't choose this alternative is because it would be just a more visual solution and wouldn't provide great accuracy, which is what we look for with our app.

3.1.2. NFC and user feedback solution

Other possibility would be to trust in the feedback of the final users. If we provide our Client app with the functionality of sending its location to the server when selected by the user, we would be able to know where each public vehicle is and an estimation of how many passenger it has. This would be possible for example by using NFC labels in each transport, when our phone reads the label, it starts sending the location for that line and vehicle every 10 seconds if the location has changed.

The problem of this solution is that it is highly technologic and user dependent. NFC is not supported by all phones yet, it is a growing technology that is not implemented at all, although it is improving quite fast. We would need the public transport approval and help to set all the NFC labels in all the public lines. And finally, another important drawback is that we must rely on the users, if we

don't have enough users with this app it will not work efficiently, as we will have some black points for some lines where there are no users travelling at that moment.

3.1.3. GPS simulation solution

For all the disadvantages previously seen on the other solutions, we must determine that the best and most logic solution is to have GPS reports for every vehicle. As we don't have access granted at the moment for this data from MPK, the creation of an app that will act as GPS will be necessary. It's thought as a "driver application", an app that could carry each bus and tram driver on his phone.

The interface is very simple and clean, we can select the number line and also a code for that line. The use of the code is to differentiate between different vehicles from the same line, as we will have several trams and buses at the same time for each line number. Once selected, it will store the line and code number as a preference, and will show it every time we run the app until we choose another one.

So if a driver starts his working day, he can run the app, press the Start button and forget about it until he finishes. The app will start sending the coordinates using the GPS provider and will show a timer with the time it has been running and will also show the number of coordinates sent at the moment.

We have used this app to make our real simulations and test the client app as we will see further in the section "3.3 Getting Coordinates".

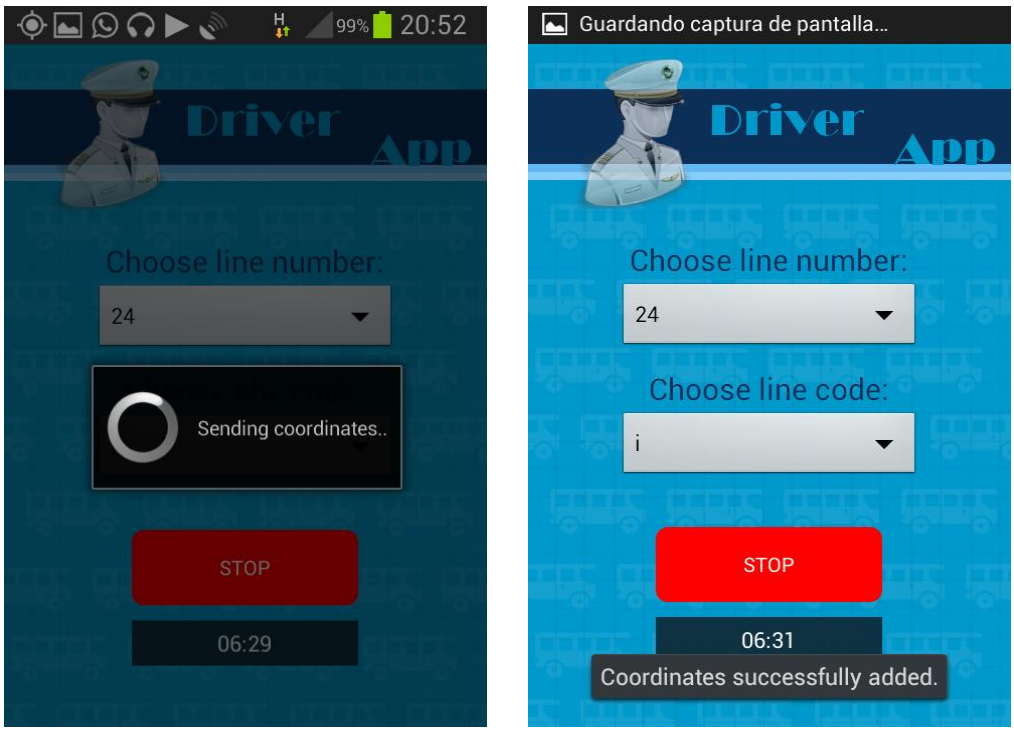


Image 3.1 – 1 – App interface while running

3.2.Functional Requirements

- We need a mobile phone with Android, minimum SDK version 8 and preferred SDK version 17.
- Our mobile phone must have GPS functionality and Network access available.
- We set a location listener that acts every 10 seconds or every 10 meters movement detected.
- The location listener gets the current coordinates from the GPS provider.
- The mobile phone must send the coordinates to a PHP service through an asynchronous task using network access to not overload the mobile's main thread.
- The PHP service will connect to our database and will store the given coordinates adding a new entry to the table "Location".
- The table Location will have the next columns:
  - **ID:** auto-increment unique identifier, primary key.
  - **Line:** line number.
  - **Code:** code number for the vehicle.
  - **Lat:** location latitude.
  - **Lon:** location longitude.
  - **Time:** timestamp that will automatically save the date and hour at which the report was created. The timestamp counts the number of seconds that have elapsed since January 1, 1970 (midnight UTC/GMT).

3.3.Getting Location

For getting the current geolocation (coordinates and bearing) we will use the classes LocationManager and LocationListener. First of all, we must give permissions to our application to grant access to location services. This is done within the AndroidManifest.xml file by adding the next lines:

```
<uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

Now, we can get a reference to the system Location Manager and define a listener that will respond to location updates:

```
locationManager = (LocationManager)
this.getSystemService(Context.LOCATION_SERVICE);

locationListenerGPS = newLocationListener() {
publicvoidonLocationChanged(Locationlocation) {
if(location != null){
EnviaCoordenadas post = newEnviaCoordenadas();
post.execute(lineArray[prefs.getInt("LINE", 0)],
codeArray[prefs.getInt("CODE", 0)],
String.valueOf(location.getLatitude()),
String.valueOf(location.getLongitude()),
String.valueOf(location.getBearing()));
        }
}
publicvoidonStatusChanged(Stringprovider,    int    status,    Bundle
extras) {}
publicvoidonProviderEnabled(Stringprovider) {}
publicvoidonProviderDisabled(Stringprovider) {}
};
```

We will use the GPS provider instead of the Network provider because of its higher accuracy and the getBearing() functionality that will allow us to know the direction in which the tram is going.

Once we have defined our listener in the onCreate() method, the only thing left is to start listening locations. We will do this when the START button is pressed by calling the Location Manager's method "requestLocationUpdates".

```
privatestaticfinalintTIME_INTERVAL = 1000 * 10; //1000 ms * 10 =
10000 milliseconds = 10 seconds
privatestaticfinalintDISTANCE = 10; // 10 meters

----------------------------------------------------------------

locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER
, TIME_INTERVAL, DISTANCE, locationListenerGPS);
```

Finally, we will stop the updates when the user presses the STOP button by calling the method "removeUpdates".

```
locationManager.removeUpdates(locationListenerGPS);
```

3.4.Sending Location

Our first step is to give permissions again to our app to grant access to Internet and Network State to check if we have an available connection or not:

```
<uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permissionandroid:name="android.permission.INTERNET"/>
```

When the system gets a new location from the GPS provider, it calls the onLocationChanged(Location location) method from our implemented listener. Within that method, we will execute an asynchronous task with the specified parameters, that will be:

- **Line** and **Code numbers.**
- **Latitude:** geographic coordinate that specifies the north-south position of a point on the Earth's surface.
- **Longitude:** geographic coordinate that specifies the east-west position of a point on the Earth's surface.
- **Bearing:** the direction angle of the vehicle. Measured in degrees. It will be used to display the direction of the vehicle on the map.

The reason of doing this within an asynchronous task is to separate the network processes from the main thread of our current app improving the efficiency of the application. Furthermore, the use of Asynchronous threads for network operations is mandatory for all the apps compiled with an SDK higher than Honeycomb (min. SDK version 8).

To send our location we will use a JSON Parser and we will send an HttpRequest with POST method to our PHP service:

```
List<NameValuePair>params = newArrayList<NameValuePair>();
params.add(newBasicNameValuePair("LINE", line));
params.add(newBasicNameValuePair("LCOD", code));
params.add(newBasicNameValuePair("LAT", lat));
params.add(newBasicNameValuePair("LON", lon));
params.add(newBasicNameValuePair("BEA", bea));

JSONObjectjson                                                    =
jsonParser.makeHttpRequest("http://transportkrakow.net/report_locati
ons.php", "POST", params);
```

This httpRequest will return a JSONObject containing the httpResponse, that we will check to see whether it was successful or not.

```php
<?php
// arrayfor JSON response
$response = array();
// checkforrequiredfields
if (isset($_POST['LINE']) &&isset($_POST['LCOD'])
&&isset($_POST['LAT']) &&isset($_POST['LON'])
&&isset($_POST['BEA'])) {

    $line = $_POST['LINE'];
    $lcod = $_POST['LCOD'];
    $lat = $_POST['LAT'];
    $lon = $_POST['LON'];
    $bea = $_POST['BEA'];

// includedbconnectclass
require_once __DIR__ .'/db_connect.php';
// connectingtodb
    $db = new DB_CONNECT();

    $result = mysql_query("INSERT INTO LOCATION(LINE, LCOD, LAT,
LON, BEA) VALUES('$line', '$lcod', '$lat', '$lon', '$bea');") ;
// checkifrowinsertedornot
if ($result) {
// successfullyinsertedintodatabase
        $response["success"] = 1;
        $response["message"] = "Coordinatessuccessfullyadded.";
echojson_encode($response);
}else {
// failedtoinsertrow
        $response["success"] = 0;
        $response["message"] = "An error occurred.";
echojson_encode($response);
}
}else {
// requiredfieldismissing
    $response["success"] = 0;
    $response["message"] = "Requiredfield(s) ismissing";
echojson_encode($response);
}
?>
```

report_locations.php

3.5.Database

We use a table “LOCATION” that stores the id, line, line code, latitude, longitude, bearing and time for each entry.

```
CREATE TABLE IF NOT EXISTS `LOCATION` (
    `ID` int(11)NOT NULL AUTO_INCREMENT,
    `LINE` smallint(6)NOT NULL,
    `LCOD` char(2) COLLATE utf8_unicode_ciNOT NULL,
    `LAT` double NOT NULL,
    `LON` double NOT NULL,
    `BEA` float NOT NULL,
    `TIME` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP
  ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (`ID`) }
```

The average size for each file is 63 Bytes. If each vehicle works from 4 am until 23 pm as average, and reports its location every 10 seconds, we have:

$$19\ \frac{hours}{day} * 60\ \frac{min}{hour} * 6 \frac{reports}{min} = 6840\ \frac{reports}{day}$$

$$6840\ reports * 65 \frac{Bytes}{report} = 444600\ Bytes$$

$$444600\ Bytes * \frac{1kByte}{1024\ Bytes} = 434{,}2\ kBytes$$

We have a total of 178 lines of public transport in Krakow (25 trams and 153 buses). If we make an estimation of 4 vehicles per line, we get that the total size of the database gets up to:

$$434{,}2\ \frac{kB}{vehicle} * 4\ vehicles * 178\ lines = 301{,}9\ \frac{MB}{day}$$

$$301{,}9\ \frac{MB}{day} * \frac{7days}{week} = \mathbf{2.113{,}3\ \frac{MB}{week}}$$

$$301{,}9\ \frac{MB}{day} * \frac{1\ month}{30\ days} = \mathbf{8{,}85\ \frac{GB}{month}}$$

Looking at this huge amount of daily entries we can work out how important is to clean the database in order to not run out of space, not decrease the efficiency and keep an acceptable response time for the queries. At this point, we would have several options. We can make weekly analysis of the data, check that the times are accurate and related to the timetables in a given range of precision and store these results in another table, “STATISTICS”. As the data stored in LOCATION will not be longer useful, we could remove the table and drop all its content weekly.

In addition, we could create a second table, “LASTLOCATION”, that would be the one connected with the client side, storing just the last location of every vehicle and improving the queries’ time response.

4. Client Side - Where is my Transport

4.1.Introduction

The chosen name for the app is “Where is my Transport?”. Its functionalities are well known as we have described them throughout this document. The main purpose is to provide the clients of public transport in Krakow with an accurate and fast tool to geolocate the different vehicles, both trams and buses in order to see their position on a map at a given time.

The user will be able to choose the desired public transport line from 3 different lists that will be shown in different dialogs: buses, trams and favorite lines.

We have tried to make a user-friendly interface, the most clean and simple possible, in order to be quick to use and accessible by all kind of public, from young people very used to new technologies, to older people that owns a smartphone but use it just to call because they are not very likely to use it in other ways.

In the next part, 4.2, we will describe the functional requirements and from section 4.3 until the end of this section, we will analyze deeply the different activities that make up the whole application and how they work.

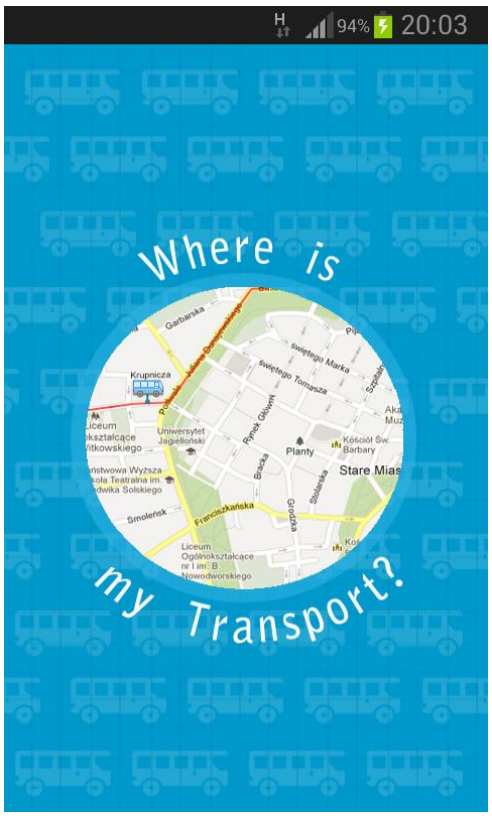
4.2.Functional Requirements

- The user can check the path of a line without internet access.
- The user can select favorite lines that will be available in a different dialog, in order to access them quickly.
- The user can remove lines from favorites unchecking the favorites star or removing all of them from the configuration menu.
- The user can hide the intro animation to have a faster access to the app.
- The user will need Internet access to receive the location of the desired public transport.
- The user will need Internet access to check the timetables, which will be linked from the Map view to the official MPK website.
- The system will inform the user with a Toast message whether it cannot get the desired location because of the lack of Internet access.
- The system will inform the user with a Toast message when there is no info available for the desired line or if there is a problem hanging the request.
- The php file must get just the last report for the desired line for each different vehicle distinguished by its particular line code.
- The app must be available in different languages: English, Polish, Spanish, Italian, French, German, and Portuguese.

4.3.Starting/Animation Activity

The App's icon is the map of Krakow with the initials WimT (Where is my Transport) on it and 2 markers representing a bus and a tram.

When we launch the application for the first time, we will see a little animation with the app's name and a little design that fades in and out. As we will see later, the user can disable this animation from the configuration menu in order to have a faster access to the app.



4.4.Selection Activity

This will be the main activity of our project. Here, the user will be able to look for a desired bus or tram. It has been designed using Android Linear, Relative and Frame Layouts.

The user will press the different areas of the screen in order to search a bus, search a tram, display the favorite lines or access to the options. When the user wants to look for a bus, he presses on "Search Bus" and the app displays a custom dialog with a list of all the available buses in Krakow and a star button to add lines to favorites or remove them from there.

When the user presses the favorite star button, it gets filled with yellow color and it is saved into the user preferences. The way of saving these preferences is using the Android Shared Preferences Class.

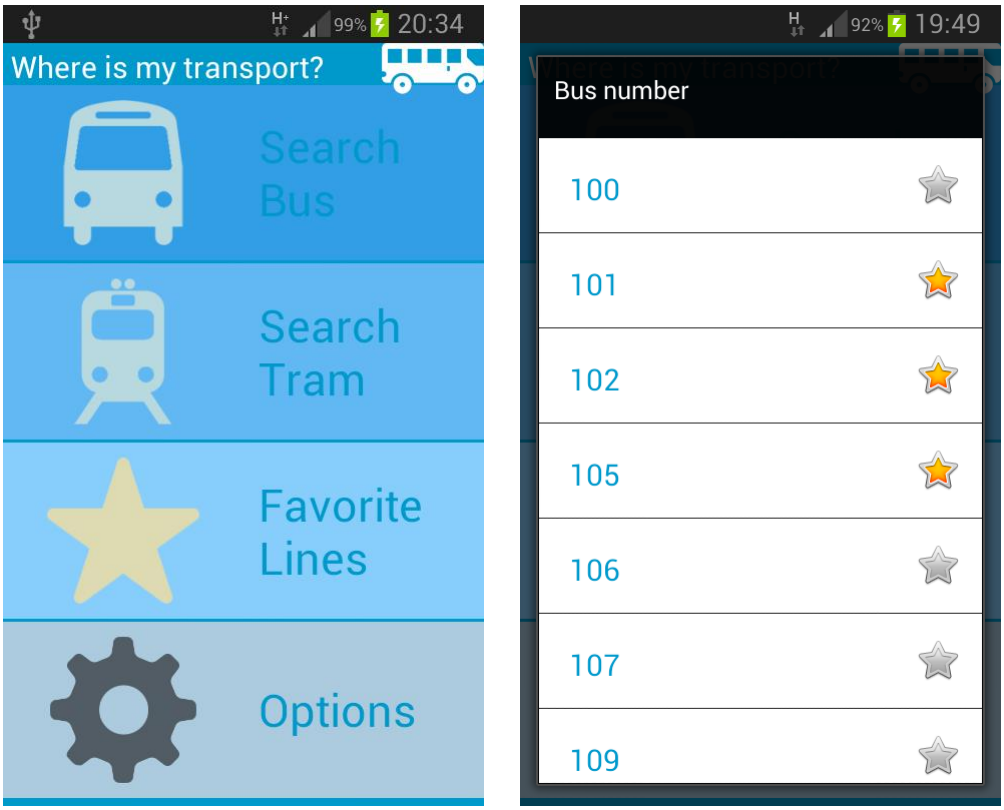


Image 4.4 – 1 – Selection Activity and bus dialog

To make a custom ListView in Android, we must create 2 layouts, one that will contain the ListView (layout_list.xml) and another for setting the layout of each row (layout_row.xml). In our case, each row has a TextView and a CheckBox Button.

We create the dialog programmatically and set the layout_list as the dialog ContentView. After that, we must use a custom ArrayAdapter and set it to our listView.

```
finalDialogdialog = newDialog(SelectionActivity.this);
dialog.setContentView(R.layout.layout_list);

ListViewlistView                    =                    (ListView)
dialog.findViewById(R.id.selectionList);
//We set ourownarrayAdapter
ArrayAdapter<Model>     ad     =     newInteractiveArrayAdapter(this,
getCustomModel(case_BUS));
listView.setAdapter(ad);
listView.setOnItemClickListener(newOnItemClickListener() {
  …
  …
  …
});
dialog.show();
```

Our custom ArrayAdapter receives as parameters the context of the application, and a List that contains all the lines and whether they are favorites or not.

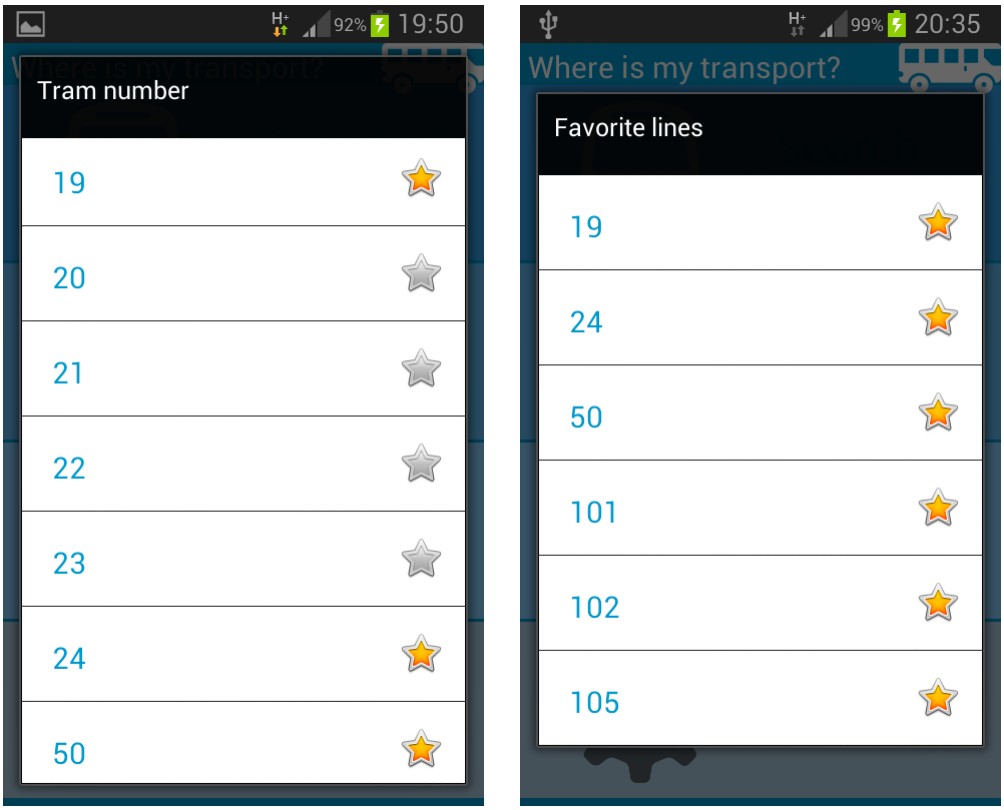


Image 4.4 – 2 – Tram dialog and favorites dialog

4.5.Configuration Activity

This is a very simple activity where the user can choose to hide the starting animation and also delete all the favorite lines saved at that moment.

We also show a little box with information about the author of the app, Iván Garrido Gómez, and the university and year.

We thought about providing the users with the possibility of changing the time interval for updates of the public transport location, but as we will see later in the next section, that would have a very bad effect on the mobile's battery life so we have dismissed this option.

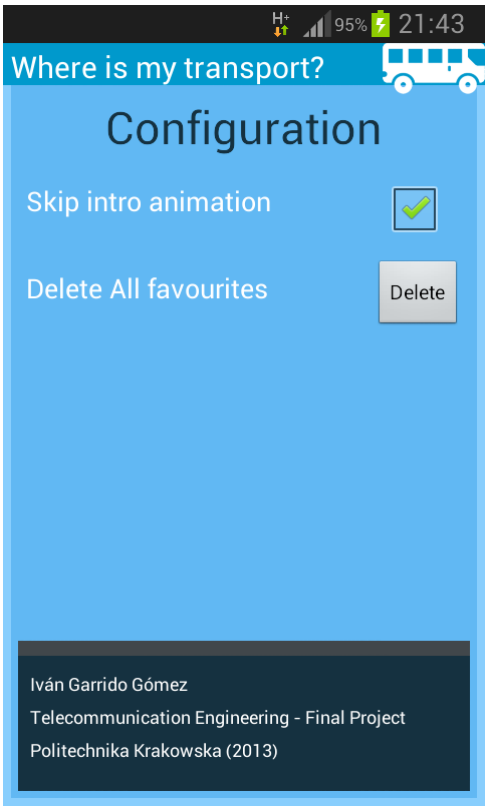


Image 4.5 – 1 – Configuration Activity

4.6.Map Activity

This activity is the most important one, and it is where all the public transport information will take place.

When we select a line, bus or tram, this activity will start and it will display the path (as a red line) and the stops (as red points) for the given line. The path will be automatically centered in the map and the user won't need internet access to display it. Internet access will only be necessary to show the map in case it is not loaded within the mobile cache.

The screen will be divided in two parts. The main one is the already commented map, where we will be able to look at the line paths, buses or trams positions, etc. The second one is a toolbar on the top, displaying the line number, and two buttons, one for looking up the timetables and another to locate the vehicles for this line.

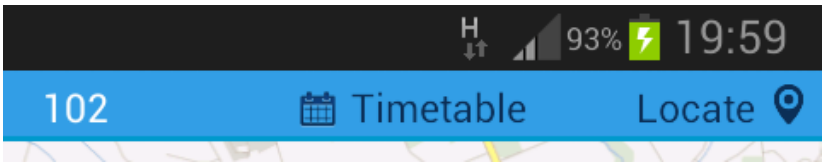


Image 4.6 – 1 – Toolbar on top

If we press the Timetable button, it will launch the browser displaying the official MPK timetable for this line on its website:

Image 4.6 – 2 – MPK line timetable

When we press the “Locate” button, the app will try to connect with the PHP service hosted in our server that will make the concrete query to the database, and will give us the result with the last location stored for the desired line.

The way of doing this, as we have already seen in the Server side (Location Sender App), will be with an Asynchronous task and a custom JSON parser that will take the HttpResponse after making the corresponding HttpRequest.

```
//Configurationforsendingthehttprequest
privatestaticfinalStringHTTP_URL                                    =
"http://transportkrakow.net/get_locations.php";
privatestaticfinalStringHTTP_GET = "GET";

-----------------------------------------------------------------------

//BuildingParameters
List<NameValuePair>params = newArrayList<NameValuePair>();
params.add(newBasicNameValuePair(TAG_LINE, intentNumber));

//Gettinglast positions bymaking HTTP request (GET type)
JSONObjectjson   =   jsonParser.makeHttpRequest(HTTP_URL,   HTTP_GET,
params);
```

Little extract from the php file to show the query and how we construct the response into a json.

```
$lid = $_GET['lid'];

$result = mysql_query("SELECT * FROM
(SELECT LINE, LCOD, LAT, LON, BEA, CONVERT_TZ( TIME, '+00:00',
'+07:00' ) AS 'TIME'
            FROM LOCATION
            WHERE LINE = $lid
            ORDER BY TIME DESC) L GROUP BY L.LINE, L.LCOD
            ORDER BY L.LCOD;");

if (!empty($result)) {
if (mysql_num_rows($result) > 0) {
// looping throughallresults
$response["lines"] = array();
while ($row = mysql_fetch_array($result)) {
// tempuserarray
$line = array();
$line["LINE"] = $row["LINE"];
$line["LCOD"] = $row["LCOD"];
$line["LAT"] = $row["LAT"];
$line["LON"] = $row["LON"];
$line["BEA"] = $row["BEA"];
$line["TIME"] = $row["TIME"];

// push single productinto final response array
array_push($response["lines"], $line);
}
// success
$response["success"] = 1;
// echoing JSON response
echo json_encode($response);
```

get_locations.php

As long as the Asynchronous connection task is taking place, we display a progress dialog informing the user about the operation and asking him to wait. It shouldn't take more than 1 or 2 seconds. When finished, the dialog will disappear and the markers will be drawn on the map. When we click on a marker, it displays the interval of time since the location was reported.

The bearing from each report will be used to display the marker in one way or the other, so the user can understand in which direction the vehicle is going just by looking at the direction that the marker is pointing at.

Image 4.6 – 3 – Bus and tram markers

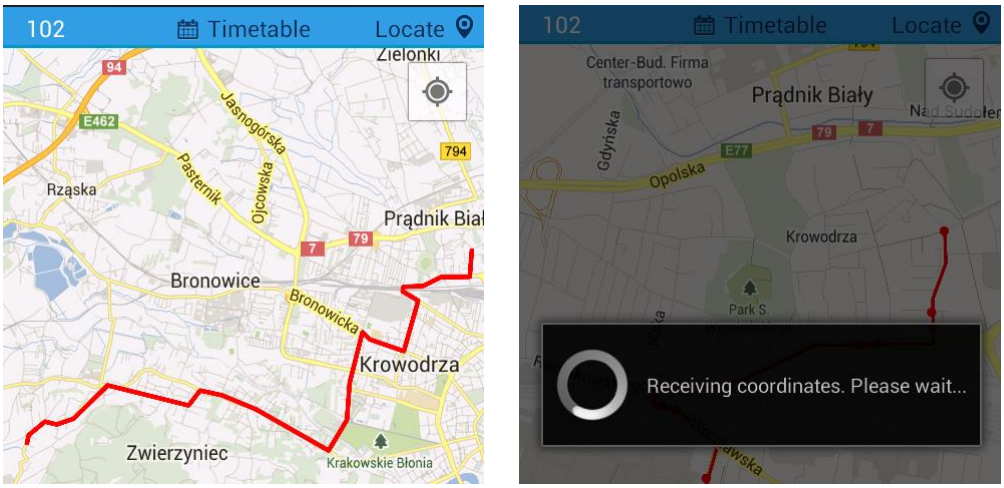


Image 4.6 – 4 – Displaying path and receiving coordinates

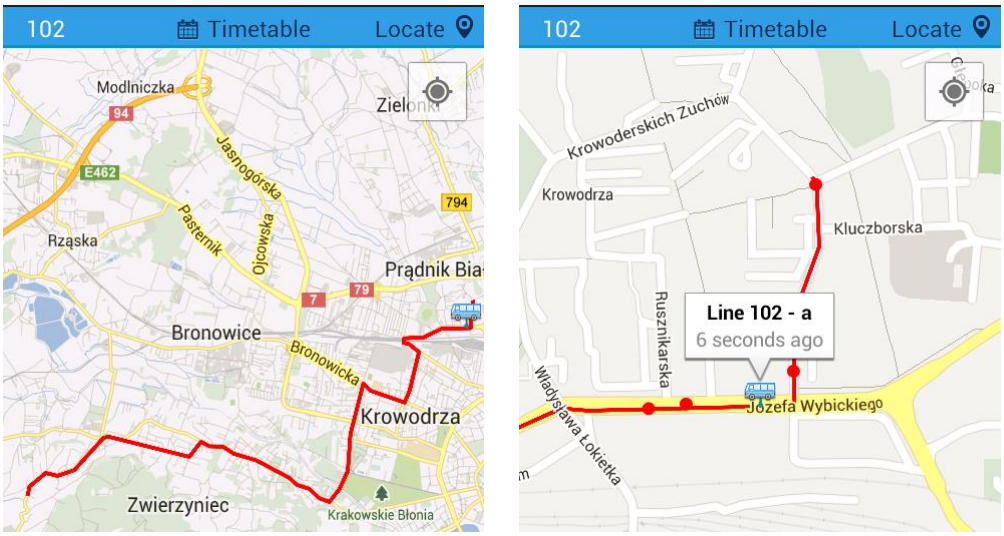


Image 4.6 – 4 – Displaying the bus marker and time information

If we are looking for a tram, we will display a tram icon instead of the bus icon previously showed:

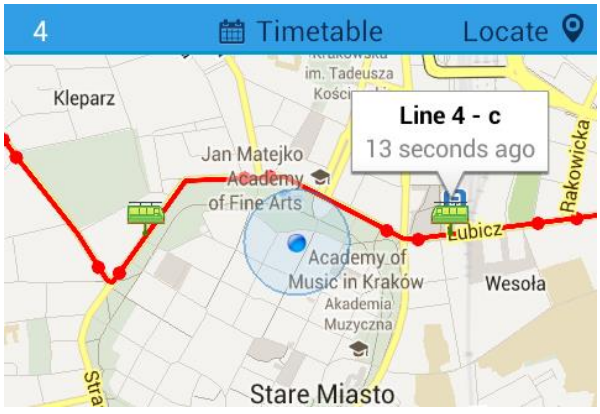


Image 4.6 – 5 – User feedback, informing the user about errors

In case there is any error, we will inform the user about it with a Toast message. The possible errors are: that we don't have internet access (or it is not working properly) or that there is no information available for that line in our database (or the database is not working at the moment).

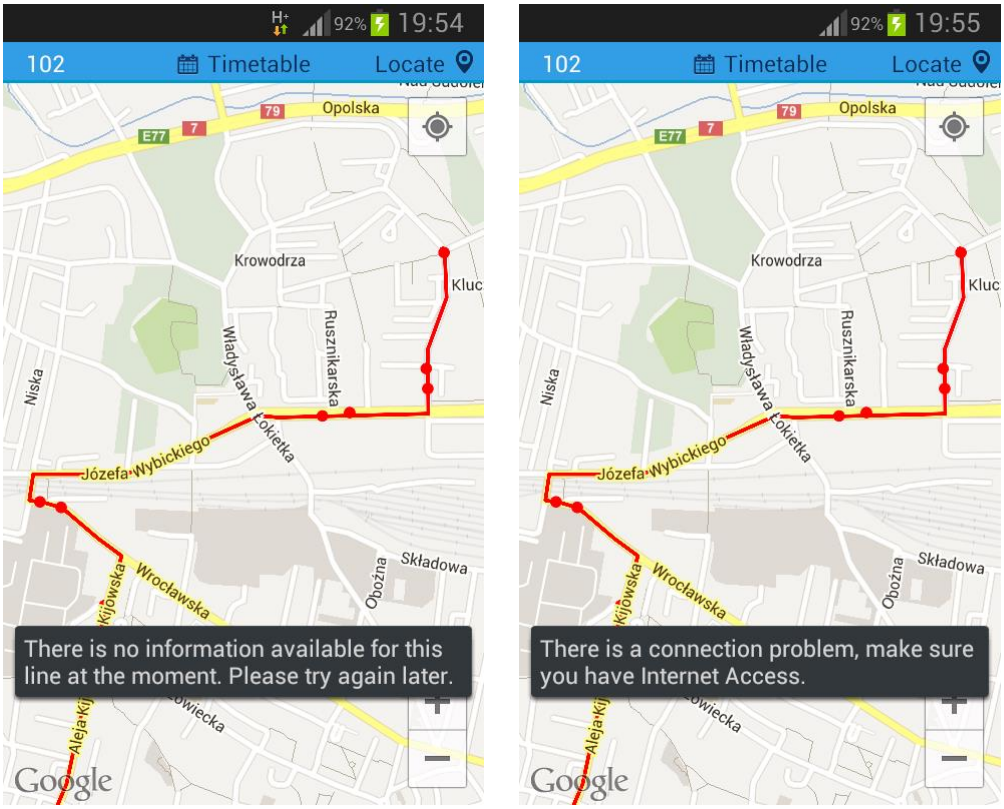


Image 4.6 – 6 – User feedback, informing the user about errors

Now we will explain how we manage the paths and stops in our application. For drawing a path, we just need to link points with a stroke in the map, the only thing we need to do is to store the coordinates of each point that forms the route, given by pairs of latitude and longitude. For the stops, it is even easier; we just need their coordinates to draw a big point on each of them.

We have different ways for storing the coordinates; the first one would be to use a sqLite database, we would need to prepopulate the database with all the points when the user opens the app for the first time.

The second possibility would be to model this information through xml files stored in the assets folder. This is the option we have chosen for one main reason: the way in which we have got the path's coordinates. We have drawn each path on google maps and exported the ".kml" file that contains all the coordinates. After that, we have created a parser that takes the relevant information from this KML file and saved it into a defined XML file:

```
<kmlxmlns="http://earth.google.com/kml/2.2">
<Document>
<name>110</name>
<Placemark>
<Point>
<coordinates>20.033474,50.075119,0.000000</coordinates>
</Point>
</Placemark>
<Placemark>
<LineString>
<tessellate>1</tessellate>
<coordinates>
        20.033470,50.075119,0.000000
        20.032780,50.075600,0.000000
</coordinates>
</LineString>
</Placemark>
</Document>
</kml>
```

KML file example

```
<?xmlversion="1.0"encoding="UTF-8"?>
<Service>
<Type>bus</Type>
<Line>110</Line>
<Center>50.09774559821183,20.09468212723732</Center>
<Zoom>11.758697</Zoom>
<Points>
<Stop>50.074815,20.034885</Stop>
<Stop>50.075448,20.038490</Stop>
</Points>
<Coordinates>
<coord>50.075119,20.033474</coord>
<coord>50.075119,20.033470</coord>
</Coordinates>
</Service>
```

XML file example

Drawing the path and stops for each line was a tedious process. We had to do it manually for each of the 153 bus lines and the 25 tram lines. It was slightly different for buses and trams just for getting the coordinates:

1) Buses: draw the path on Google Maps and, export the kml file.
   Trams: draw the path with a web service polyline generator.
2) Parse that file obtaining the coordinates into our XML.
3) Get the coordinates for each stop in that line with a web service that gives you the coordinates from the point where you clicked.
4) Put these stops into our XML file.

5) Get the coordinates and the zoom of the camera for centering the path in the map. For this, I put a listener on a button of my MapActivity, that when pressed, saved the current zoom and center position of the camera in a file in the external memory of the phone with the name "Zoom_[number].xml". So I had to center each line with the correct zoom and then press the button to store it into a file.
6) Put the information of those files into the line XML file with a parser.

Once we have finished all the XML files for each line, we create a DOM parser that will read the selected line xml file stored in the assets and that will process all the information it contains: coordinates for the path, coordinates for the stops, center coordinates and level of zoom.

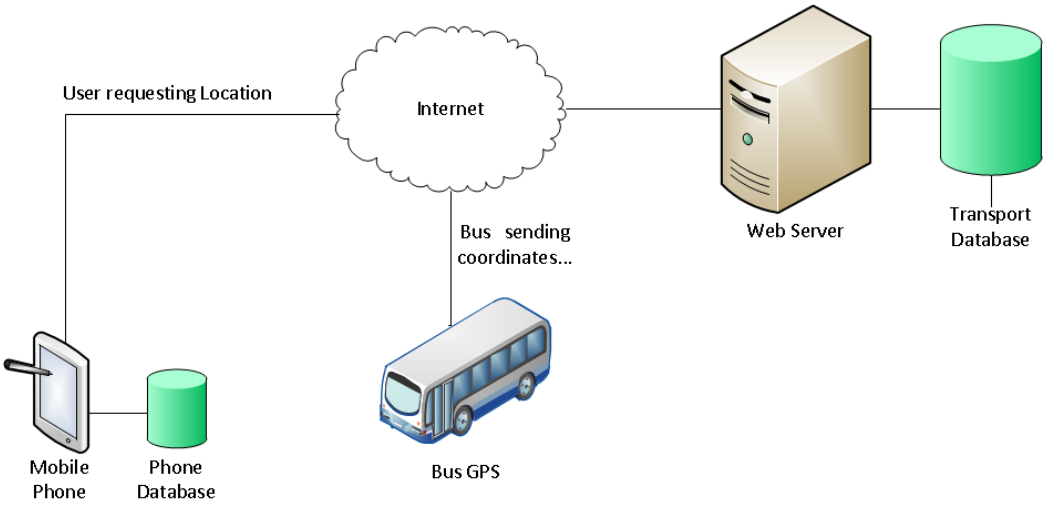
We will use Google API v2 for all the things related with drawing in the map, as explained below in section number 7.

4.7.Multilingual availability

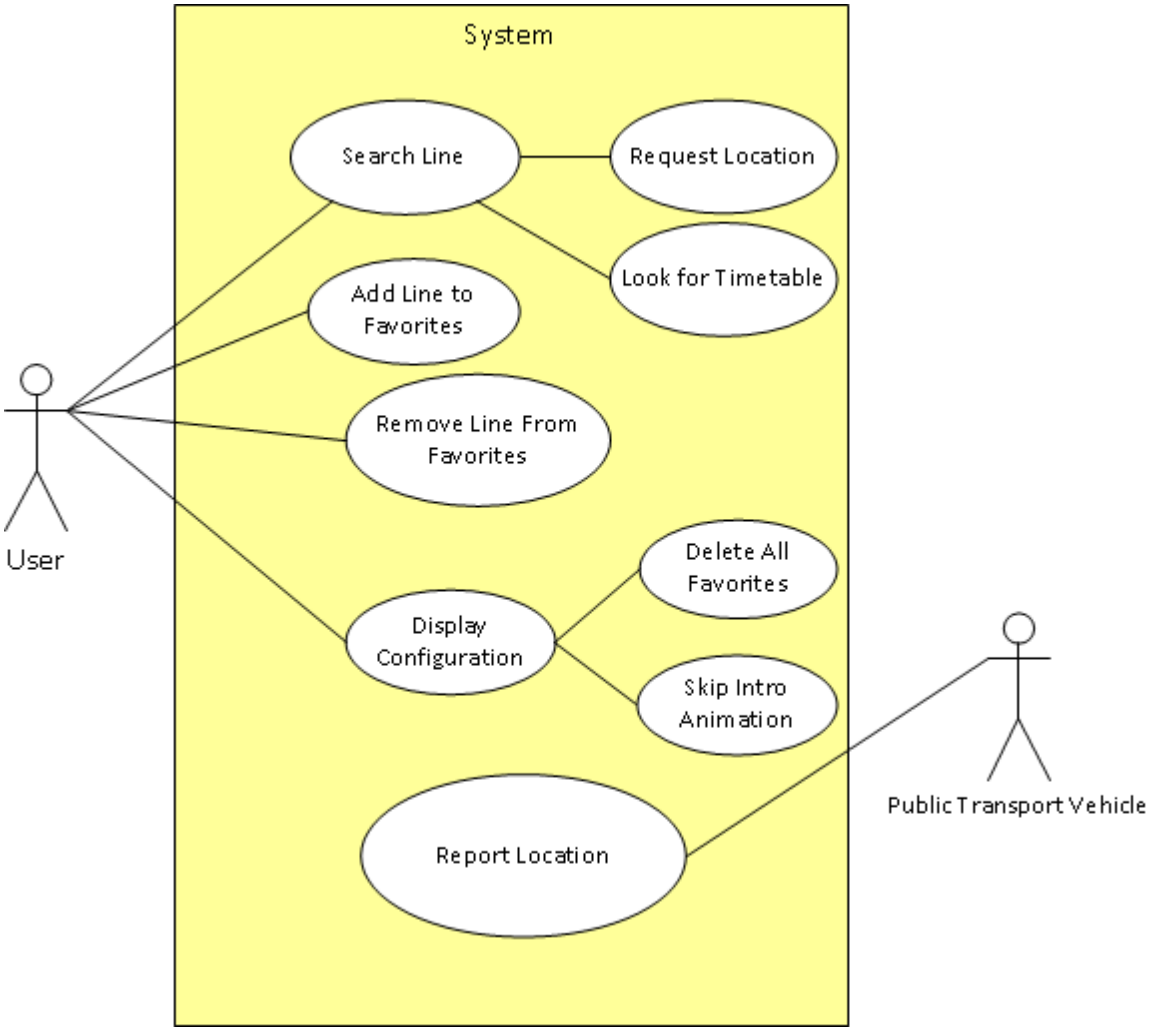
Thanks to the Android easiness for making a multilingual app, we have translated the app in 7 different languages: English, Polish, Spanish, Portuguese, German, Italian and French. Android will automatically take the values from the language we have selected in our mobile phone preferences.

We just need to create a new "values" folder with the code of the country, and put inside it a new strings resource with the strings in the desired language. For example, "values-pl/strings.xml" will contain all the strings in Polish version, "values-es/strings.xml" in Spanish, etc.

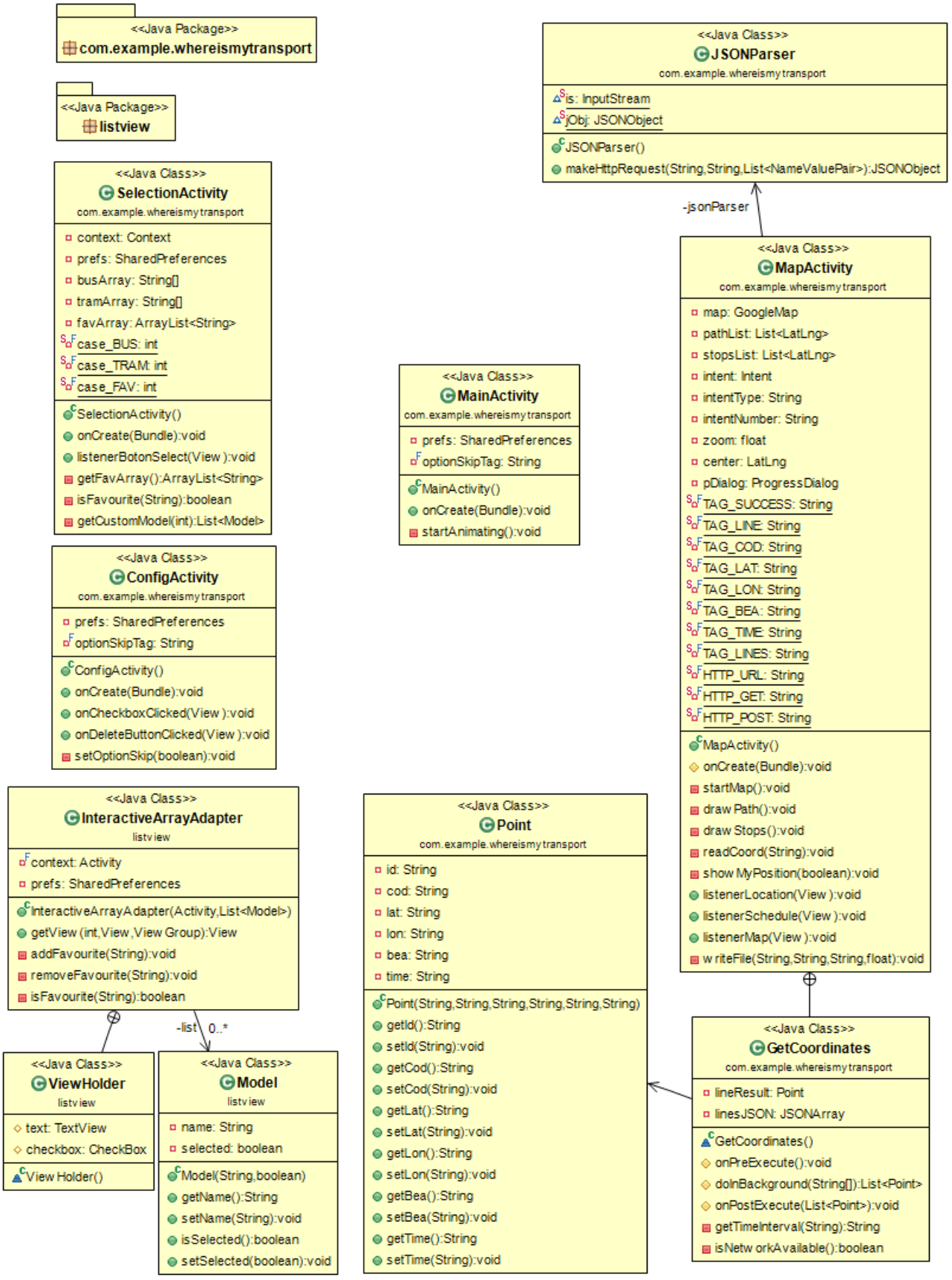
4.8.General Diagram



4.9.Use Case Diagram



4.10. Classes Diagram



5. Tests performed

Once our application was finished, we have run it in a real environment to test how it works. The steps followed have been:

1) To install the Location Sender App in one device, and go into a tram (line number 24).
2) When we were ready, we pressed the Start button and it began to send its location to our server.
3) After a while, we pressed the button Stop and it finished the oLocationChanged listener so we no longer had reports for the tram geolocation
4) At the same time, with a device running the Client App (Where is my transport) we were checking that it was recovering the accurate data from the server with the most recent location of the tram.
5) Finally, we changed the query to display all the collected data for the tram, and not just the last position, in order to trace the complete path and see it drawn in the map.

We can find below the entries that were stored in the database. As we can see in the TIME column, the reports are taken by an average interval of 10 seconds. Sometimes it takes more time and sometimes it takes a bit less, because it has 2 variables to listen the location changes, one is the min. distance (10 meters) and the second is the min. time (10 seconds), it should accomplish both requirements before getting a new location. But the precision isn't perfect and it depends on the network access also while sending the coordinates:

ID	LINE	LCOD	LAT	LON	BEA	TIME
136	24	i	50.064845625311136	19.943554736673832	132	2013-06-10 13:46:07
137	24	i	50.06484960671514	19.94443316012621	126	2013-06-10 13:46:16
138	24	i	50.06477362476289	19.944516895338893	97	2013-06-10 13:46:42
139	24	i	50.064597646705806	19.944858960807323	132	2013-06-10 13:46:50
140	24	i	50.06431991234422	19.94501620531082	174	2013-06-10 13:47:00
141	24	i	50.063717337325215	19.945096001029015	174	2013-06-10 13:47:10
142	24	i	50.063042803667486	19.94507672265172	180	2013-06-10 13:47:20
143	24	i	50.062167439609766	19.944738261401653	200	2013-06-10 13:47:29
144	24	i	50.061690718866885	19.94440734386444	193	2013-06-10 13:47:40
145	24	i	50.061115552671254	19.94403351098299	204	2013-06-10 13:47:50
146	24	i	50.06069473922253	19.94362296536565	224	2013-06-10 13:48:00
147	24	i	50.060580745339394	19.943464379757643	226	2013-06-10 13:48:10

148	24	i	50.06012397352606	19.942859038710594	229	2013-06-10 13:48:21
149	24	i	50.05987427663058	19.942584531381726	212	2013-06-10 13:48:29
150	24	i	50.05977272987366	19.94249383918941	215	2013-06-10 13:48:39
151	24	i	50.059870295226574	19.942479422315955	328	2013-06-10 13:49:11
152	24	i	50.05977553781122	19.94247472845018	190	2013-06-10 13:49:34
153	24	i	50.059448475949466	19.94236635044217	137	2013-06-10 13:49:40
154	24	i	50.05861384794116	19.943061964586377	154	2013-06-10 13:49:51
155	24	i	50.057752104476094	19.9438223708421	146	2013-06-10 13:50:00
156	24	i	50.05716570653021	19.944499880075455	144	2013-06-10 13:50:12
157	24	i	50.05693801213056	19.944833647459745	129	2013-06-10 13:50:21
158	24	i	50.056827957741916	19.945161966606975	143	2013-06-10 13:50:33
159	24	i	50.05673856474459	19.94532499462366	145	2013-06-10 13:51:39
160	24	i	50.05630475934595	19.94572858326137	144	2013-06-10 13:51:48
161	24	i	50.05576035473496	19.946366446092725	139	2013-06-10 13:52:01
162	24	i	50.054859216324985	19.947011601179838	151	2013-06-10 13:52:15
163	24	i	50.05409621167928	19.947735713794827	156	2013-06-10 13:52:21

Image 5 – 1 – Entries stored in database for test over tram line 24.

After changing the query to show all the results for the searched line, when we press the button locate it loads in the map all the reports we saw in the previous table that were stored in the database.

As we can see, it works great, sometimes the accuracy of the gps is not perfect and the marker is a bit outside the path, but most of them fit perfectly. The bearing algorithm works properly as well, and it shows the direction on the map as it should.

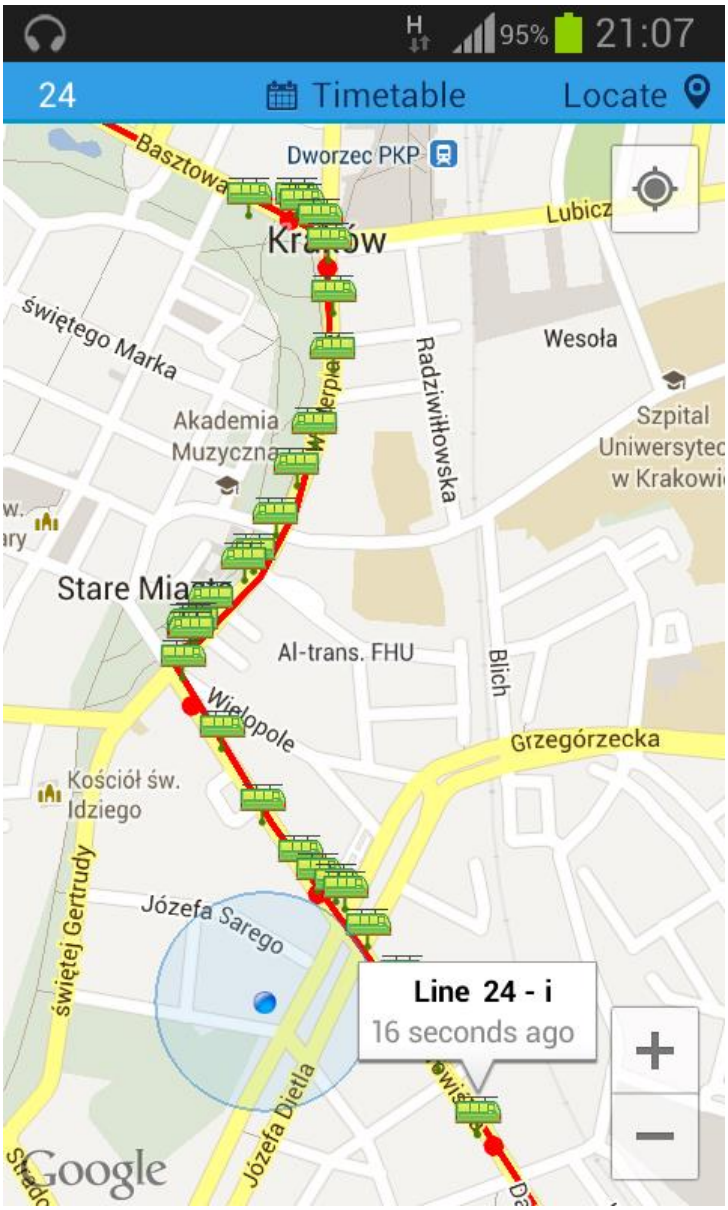


Image 5 – 2 – Displaying the markers for test over tram line 24.

6. Administration tool - Statistics

6.1.Introduction

As we have overviewed in the introduction, gathering the location reports of each public transport vehicle allows us to develop different and useful features. One of them would be the possibility of checking that all the official timetables are fulfilled.

Each vehicle is sending its location every X time. This data is stored in a database with the latitude, longitude, bearing and a timestamp, so we can find out when it is arriving at each stop and look for errors in the system of public transport timetables provided by MPK.

To achieve this, as MPK doesn't provide any kind of API or open database to check the timetables, we should create our own big database taking the data that MPK offers in the form of html tables, modeling all the trips, stops and timetables for each of them.

Just making some fast calculations, if there are 178 lines, with outbound and inbound trip each of them, with an average of 30 stops per trip, and a schedule that distinguishes around 20 hours and 3 different types (weekly, Saturdays and holidays), with an average of 5 different trips per hour, we have:

$$178\ lines * \frac{2\ ways}{line} * 30\ \frac{stops}{way} * 3\ \frac{schedules}{stop} = 32.040\ schedules$$

$$32040\ schedules * \frac{20\ hours}{schedule} * 5\ \frac{trips}{hour} = \mathbf{3.204.000\ trips}$$

As far as we can see, we would need to introduce over 3 million of different trips in our database. We can conclude that digitizing all the schedules for a municipal transport system is an industrial-scale project beyond the scope of this geolocation project.

Instead, we will make a little demonstration just for one line of how it would work and what steps we should follow. And we will explain what design we should follow to achieve the complete project, something that could be a big project itself or a future improvement and continuation of this one.

6.2.Database modeling and design

The first step for developing this software would be to create a proper database design. It should be normalized in order to minimize redundancy and dependency, which involves isolating data by creating smaller tables and defining relationships between them. We would need:

Line table	*one row for each bus and tram line*
Line (PK)	
Description	e.g. 106-inbound or 106-outbound
Station table	*one row for each stop, including ends of trips*
Line	part of PK, FK to Linetable
StationID	part of PK
Description	e.g. Bazstowa Lot Street
Lat	
Lon	
Trip table	*one row for each trip*
TripID (PK)	auto-increment number
Line	FK to Linetable
Description	e.g. 05:22 trip Bronowice to NowaHuta
Schedule table	*one row for each scheduled time for each trip at each stop*
ScheID (PK)	auto-increment number
Line	FK to Station
StationID	FK to Station
TripID	FK to Trip
Time	e.g. 12:53

Once we have designed and created the table with all the relationships described, our next step would be to insert all the data into it. As we have already said, we do not have a source of data in an easy format to parse. For the timetables we could create an html parser that takes the times for each stop in each line, but we would need also to get the coordinates manually for each public transport stop in Krakow, and to store it within the Station table.

After inserting all the data, all we need to do is to make queries that compare the coordinates of the reports from our LOCATION table with the coordinates from the STATION table, and check that they are in the range of a given distance, *e.g.* 30 meters. Now, we would just compare the time of each report with the times related to that stop and stored within the SCHEDULE table. If the time exceeds in a given time (*e.g.* 2 min) we would create a new report in a table STATISTICS that would keep track of the delays.

6.3.Haversine formula

To compare the distance between 2 different and close locations, we should use the haversine formula:

$$haversin\,(d/r) = haversin(\phi_2 - \phi_1) + \cos(\phi_1)\cos(\phi_2)haversin(\lambda_2 - \lambda_1)$$

$$haversin\,(\theta) = sin^2\left(\frac{\theta}{2}\right) = \frac{1 - \cos(\theta)}{2}$$

- ***d*** *is the distance between the 2 points.*
- ***r****is the radius of the sphere, in our case the radius of the Earth.*
- $\boldsymbol{\phi_1, \phi_2}$*are thelatitudes of point 1 and point 2.*
- $\boldsymbol{\lambda_1, \lambda_2}$*are the longitudes of point 1 and point 2.*

Applying this formula into a MYSQL query, we can obtain the distance:

```
(6371000*ACOS( COS( RADIANS(S.LAT) ) * COS( RADIANS(L.LAT) ) * COS( RADIANS(L.LON) - RADIANS(S.LON) ) + SIN( RADIANS(S.LAT) ) * SIN( RADIANS(L.LAT ) ) ) ) ASdistance
```

- *Note that the distance we are using is the radius of the Earth in meters, so the result of the equation will be also given in meters.*
- *S.LAT and S.LON are the stop latitude and longitude.*
- *L.LAT and L.LON are the location record latitude and longitude.*

6.4.Demonstration

We have run a little demonstration over a php web service of how would be an administration tool for public transport employees that could login and check the proper system functioning and the vehicles delays over the official MPK timetables.

To achieve this, we have made a simpler model of database. We have created 3 additional tables, one for stops, a second one for timetables and a third one to store the statistics reports for delayed lines.

As the purpose of this demonstration is just to show how the queries would work, we have just inserted the data related to bus line number 100.

LID	NAME	LAT	LON	LINE
1	Kopiec Kościuszki	50.054153	19.893207	100
2	Aleja Waszyngtona	50.053171	19.896211	100
3	Aleja Waszyngtona	50.053207	19.897284	100
4	Malczewskiego	50.049597	19.903206	100
5	Malczewskiego	50.048523	19.903979	100
6	Salwator	50.053676	19.917669	100

Image 6.4 – 1 – STOPS table

TTID	HOUR	WEEKLY	SATURDAY	HOLIDAYS	LID
27	7	03	03	03	4
28	8	03	03	03	4
29	9	03	03	03	4
30	10	03	03	03	4
31	11	03	03	03	4
32	12	03	03	03	4
33	13	03	03	03	4
34	14	03	03	03	4
35	15	03	03	03	4
36	16	03	03	03	4
37	17	03	03	03	4
38	18	03	03	03	4
39	19	08	0858	0858	4

Image 6.4 – 2 – TIMETABLE table

** Note that TIMETABLE.LID is FK of STOPS.LID.*

We will make the following query to obtain the reports that are very close to the stops, in our case just the reports that are for the searched line and in a range of 20 meters max from the stop:

```
SELECT S.LID, S.NAME, L.LINE, L.LCOD, L.TIME,
MINUTE( CONVERT_TZ( L.TIME,  '+00:00',  '+07:00' ) ) AS  'MIN',SECOND( CONVERT_TZ(
L.TIME,'+00:00','+07:00' ) ) AS  'SEC',
(  6371000  *ACOS(COS(RADIANS(S.LAT)  )  *COS(  RADIANS(L.LAT)  )  *  COS(
RADIANS(L.LON) - RADIANS(S.LON) ) + SIN(RADIANS(S.LAT) ) * SIN(RADIANS(L.LAT ) ) ) )
AS distance, T.WEEKLY, T.SATURDAY
FROM LOCATION L, TIMETABLE T, STOPS S
WHERE L.LINE = $line
AND T.HOUR=(HOUR( CONVERT_TZ( L.TIME,'+00:00',  '+07:00' )))
AND S.LID=T.LID
HAVING distance <20
ORDER BY distance;
```

We will compare the time from the obtained reports with the time at which it was supposed to be from the timetables of that stop (identified by its LID). And if it exceeds in more than 90 seconds we will consider that vehicle as delayed and will report it by inserting a new row in our table STATISTICS:

```
INSERT INTO STATISTICS(`LINE`,`LCOD`,`NAME`,`TIME`,`DIF`)VALUES
('$line','$lcod','$name','$timestamp', '$dif');
```

Where “DIF” is the interval in seconds between the expected timetable and the time from the report.

In our demo, we will add the following reports to the LOCATION table:

ID	LINE	LCOD	LAT	LON	BEA	TIME
167	100	a	50.05318136302678	19.89604577422142	6	2013-06-12 10:02:35
168	100	a	50.04966109219669	19.90317240357399	89	2013-06-12 10:03:26
169	100	a	50.05003999871856	19.902829080820084	24	2013-06-12 18:41:56
170	100	a	50.05273014900257	19.915026426315308	24	2013-06-12 18:41:56

Image 6.4 – 3– Added reports for simulation at LOCATION table

And after running the script we will get that there is one report delayed that will be inserted in our STATISTICS table. *(Note that the timestamp is stored in USA UTC Time Zone, we will convert it to European TimeZone every time we work with the data)*.

ID	LINE	LCOD	NAME	TIME	DIF
1	100	a	Aleja Waszyngtona	2013-06-13 08:59:50	95

Image 6.4 – 4 – STATISTICS table

6.5.Interface

As mentioned already, the interface and the way of accessing to the database are through PHP scripts. Here we will show just the visual interface made on HTML, PHP and CSS. You can find the code in the resource files attached to this document.

http://www.transportkrakow.net/admin/

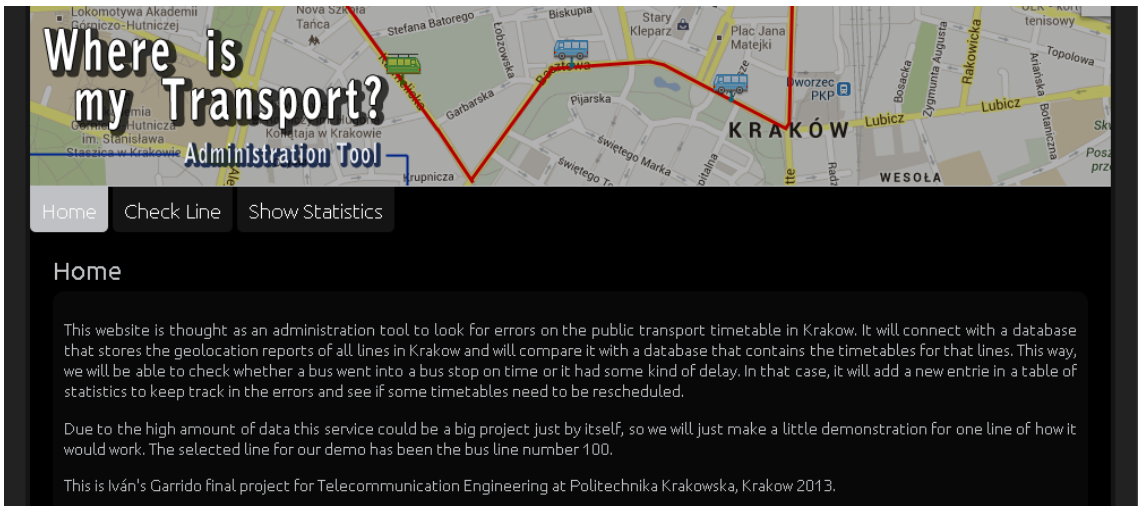


Image 6.5 – 1 – Home webpage

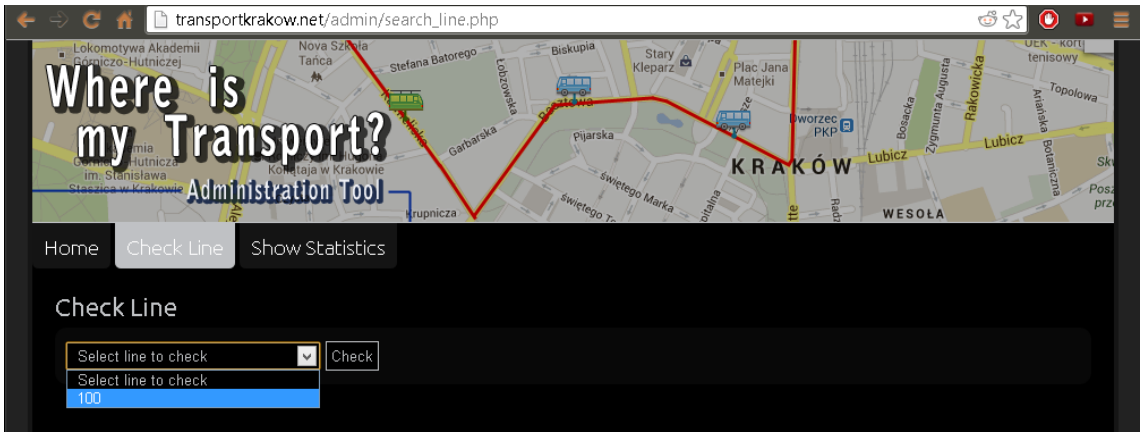


Image 6.5 – 2 – Check webpage

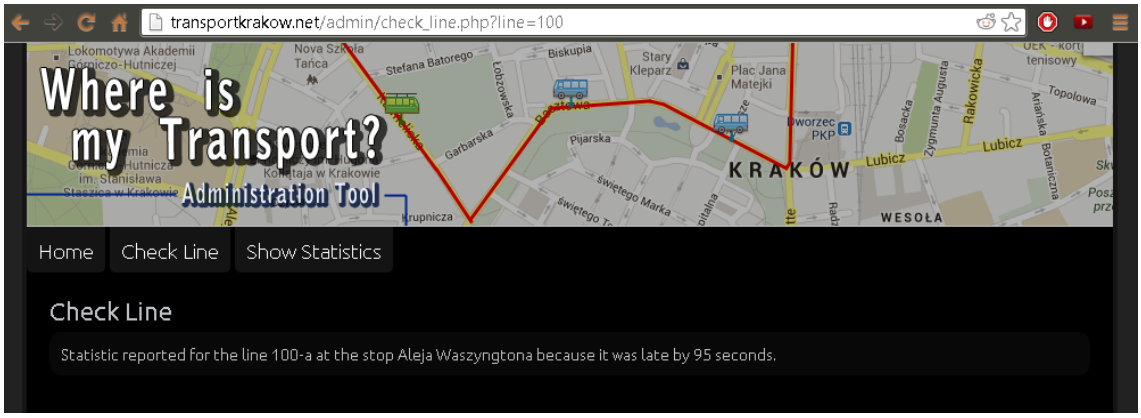


Image 6.5 – 3 – Check result webpage

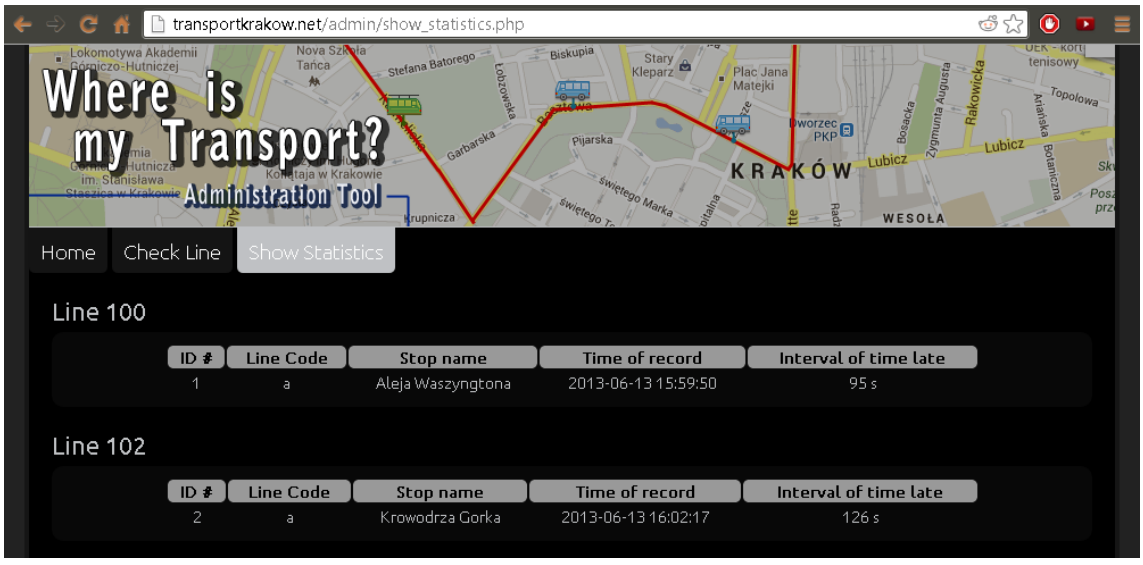


Image 6.5 – 4 – Show statistics webpage

7. Google's API, a closer look

For the development of all the map related elements we have used Google's Map API v2. At the beginning we started using Google's Map API v1, but as Google deprecated it and stopped providing support we decided to migrate to v2.

Google released Map API v2 with the next improvements:

- Because maps are encapsulated in the MapFragment class, you can implement them by extending the Android standard Activity class, rather than extending the MapActivity used in version 1.
- The Maps API now uses vector tiles. Their data representation is smaller, so maps appear in your apps faster, and use less bandwidth.
- Caching is improved, so users will typically see a map without empty areas.
- Maps are now 3D. By moving the user's viewpoint, you can show the map with perspective.

All this improvements make the app migration between APIs totally worthy.

We can find one big difference for drawing paths/stops on the map between the 2 versions. While in v1 we should do it manually creating a new custom overlay, linking GeoPoints within the draw methodusing path, canvas and paint tools and finally adding this overlay to our map; polylines and circles shapes in v2 make things much easier.

```
publicbooleandraw(Canvascanvas,     MapViewmapView,     booleanshadow,
longwhen) {
      //Wecreatethepath and thepointsthatwillperformit.
      Pathpath = newPath();
      Point pA = new Point();
      for (int i = 0; i <_points.size(); i++) {
      //Gettingtheprojectionto set thecoordinates as pixels
      mapView.getProjection().toPixels(_points.get(i), pA);
            if (i == 0) path.moveTo(pA.x, pA.y);
            elsepath.lineTo(pA.x, pA.y);
            //Wedraw a new pathevery 5 points
            if(i%5==0 || i == _points.size()-1){
                  Paintpaint = newPaint();
                  paint.setAntiAlias(true);
                  paint.setColor(_pathColor);
                  paint.setStyle(Paint.Style.STROKE);
                  paint.setStrokeJoin(Paint.Join.ROUND);
                  paint.setStrokeCap(Paint.Cap.ROUND);
                  paint.setStrokeWidth(mapView.getZoomLevel()-10);
                  paint.setAlpha(_alpha);
                  if (!path.isEmpty()) canvas.drawPath(path, paint);
                        path = newPath();
                        path.moveTo(pA.x, pA.y);
                  }
```

```
                }
        }
        returnsuper.draw(canvas, mapView, shadow, when);
}
```

Drawing a path with Google v1 API

```
privatevoiddrawPath(){
        PolylineOptionsoptions = newPolylineOptions();
        options.width(4);
        options.color(Color.RED);
        for(int i = 0; i<pathList.size(); i++ ){
                options.add(pathList.get(i));
        }
        map.addPolyline(options);
}
```

Drawing a Path with Google v2 API

```
privatevoiddrawStops(){
        CircleOptionsoptions = newCircleOptions();
        options.radius(3);
        options.strokeColor(Color.RED);
        options.fillColor(Color.RED);
        for(int i = 0; i<stopsList.size(); i++ ){
                options.center(stopsList.get(i));
                map.addCircle(options);
        }
}
```

Drawing the stops with Google v2 API

8. Future improvements

In the world of Android applications we can conclude that one of the most important keys to success is to provide continuous support. If we just forget about them without updating content, fixing problems and making some little changes, it's highly probable that our competitors will win the race and our application will rot in oblivion. There is always a little chance to improve everything, and with our app it is not different.

An extra functionality we could add in the future would be the possibility of showing the last location of different lines at the same time (both buses and trams). Sometimes we can reach the same destination by different lines, and this way we could see all of them together.

We could set a new user preference to limit the search depending on the current user location. *E.g*. just showing the public transports that are within 1.5 km radius from the user location.

We could also make an internal database for the timetables to not need internet while checking them. Internet would just be needed to update it from time to time and to use the location functionality.

9. Bibliography and tools

Bell, Chris. "Google maps polylines code generator." 17/04/2013. <http://www.doogal.co.uk/polylines.php>

Google Developers. "Google Maps API v2 for Android." Updated on 26 February 2013. 15/05/2013. <https://developers.google.com/maps/documentation/android/?hl=fr-FR>

Iconfinder. "Provider of free icons." <http://www.iconfinder.com/>

Opción 5 Desarrollos tecnológicos. "Web service for getting coordinates by clicking." 21/04/2013. <http://www.opcion5.com/soporte-hosting/latitud-y-logitud>

Pizzetti, Franchesco. "Geolocation in Public Transportation - Passenger Security." 5 June 2006. 02/06/2013. <http://www.garanteprivacy.it/web/guest/home/docweb/-/docweb-display/docweb/1672796>

Tamada, Ravi. "How to connect Android with PHP, MYSQL." 21/05/2013. <http://www.androidhive.info/2012/05/how-to-connect-android-with-php-mysql/>

Stackoverflow forums. "Checking and solving Android and MYSQL doubts." 23/05/2013. <http://stackoverflow.com/questions/tagged/android>

Stackoverflow forums. "MySQL Great Circle Distance (Haversine formula)." 07/06/2013. <http://stackoverflow.com/questions/574691/mysql-great-circle-distance-haversine-formula>

Wikipedia. "Haversine formula". Updated on 6 May 2013. 07/06/2013. <http://en.wikipedia.org/wiki/Haversine_formula>

Meier, Reto. "A deep dive into location." 25/05/2013. <http://android-developers.blogspot.com/2011/06/deep-dive-into-location.html>